Experiment 5

Name: Ansh Chughria

Roll: 20

Batch: T11

Aim: To Build the pipeline of jobs using Maven / Gradle / Ant in Jenkins, create a pipeline script to Test and deploy an application over the Tomcat server

Theory:

Continuous Integration (CI) involves automatically testing and building software in a shared repository, allowing developers to frequently integrate their work. In this context, Jenkins is used as the CI server to build, test, and deploy an application. The pipeline you will set up will utilize **Maven**, **Gradle**, or **Ant** to automate the process, and **Tomcat** will serve as the application server for deployment.

1. Understanding Jenkins Pipeline

A **Jenkins pipeline** is a series of automated processes that include building, testing, and deploying software. These processes are defined as **jobs** in Jenkins. A pipeline script can be created in **Jenkinsfile**, which can be version-controlled along with the source code.

Jenkins offers two types of pipelines:

1. **Declarative Pipeline**: A more structured and easier-to-understand format, ideal for most use cases.
2. **Scripted Pipeline**: Offers more flexibility but requires knowledge of Groovy scripting. It is more suited for complex workflows.

In this case, we will use a **Declarative Pipeline** script that will:

- Build the application using **Maven**, **Gradle**, or **Ant**.
- Run tests to ensure the application is working as expected.
- Deploy the application to a **Tomcat** server.

---

2. Jenkins Pipeline Stages

A typical Jenkins pipeline for building, testing, and deploying an application consists of the following stages:

a. Build Stage

In this stage, Jenkins will use **Maven**, **Gradle**, or **Ant** to compile and package the application. The choice of tool depends on the project's build system.

- **Maven**: Uses the pom.xml file to manage dependencies and build lifecycle.
- **Gradle**: Uses build.gradle to define tasks and dependencies.
- **Ant**: Uses build.xml to define tasks and configurations

---

b. Test Stage

After building the application, the next stage involves running tests to ensure the application behaves as expected. This can include unit tests, integration tests, or other forms of testing depending on the setup of your project.

- **Maven**: Run tests using the mvn test command.
- **Gradle**: Run tests using the gradle test command.
- **Ant**: Run tests using ant test.

c. Deploy Stage

The final stage of the pipeline is the **deployment** phase. In this phase, Jenkins will deploy the built application to a **Tomcat server**.

Tomcat is a popular open-source Java web server and servlet container, used to deploy web applications. Jenkins can deploy the application by copying the generated WAR file to the Tomcat webapps directory, where it will be automatically deployed.

Example for Deploying to Tomcat:

1. **Install Tomcat Plugin in Jenkins** (Optional but recommended):
    - Jenkins provides a **Deploy to Container** plugin, which allows you to deploy your applications to a **Tomcat** server directly from Jenkins. To install it:
        - Go to **Manage Jenkins > Manage Plugins**.

- Search for **Deploy to Container** plugin and install it.

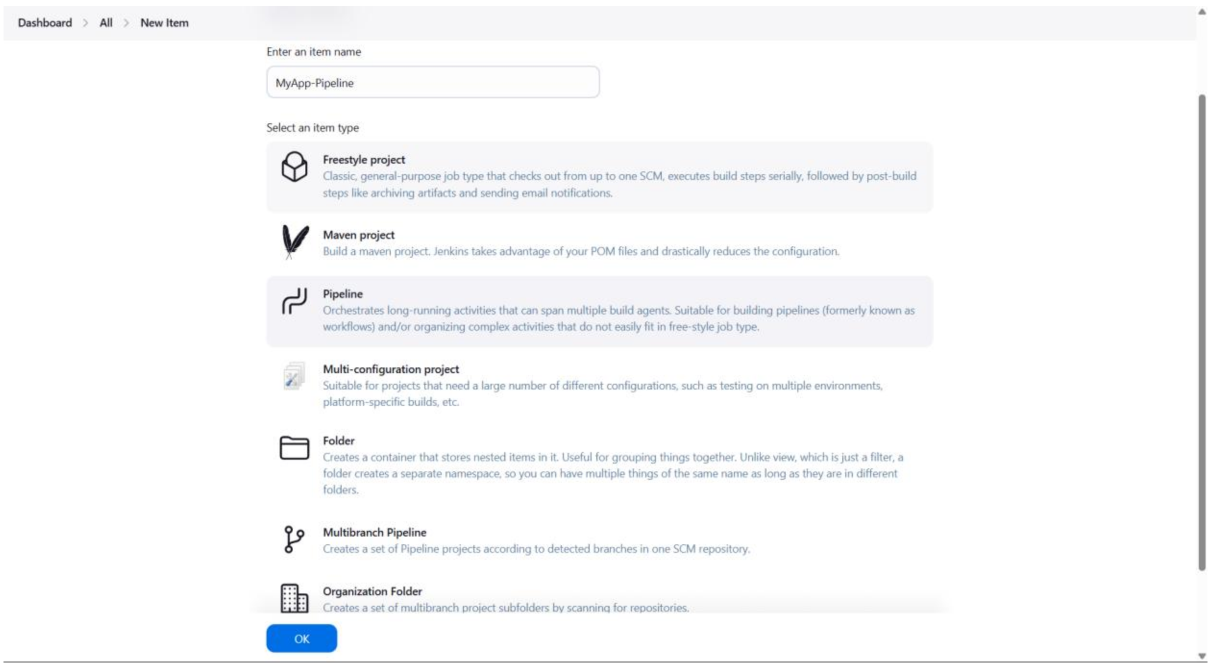
2. **Configure Deployment Credentials**:
   - In Jenkins, configure the Tomcat server's deployment credentials (username, password) under **Manage Jenkins > Configure System**. Enter the URL of your Tomcat server (e.g., http://localhost:8080/manager/text) and the credentials for the Tomcat Manager application.
3. **Deploy WAR File**:
   - After configuring Jenkins, you can use the following code in your pipeline to deploy the WAR file to Tomcat.

4. Configuring Tomcat for Automatic Deployment

For **automated deployment** to **Tomcat**, ensure that:

- **Tomcat Manager** is enabled and configured to accept deployment commands (for example, via HTTP requests).
- You provide the correct **Tomcat Manager credentials** (username and password) to Jenkins for deployment.

Implementation:


Dashboard > All > New Item

Enter an item name

MyApp-Pipeline

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

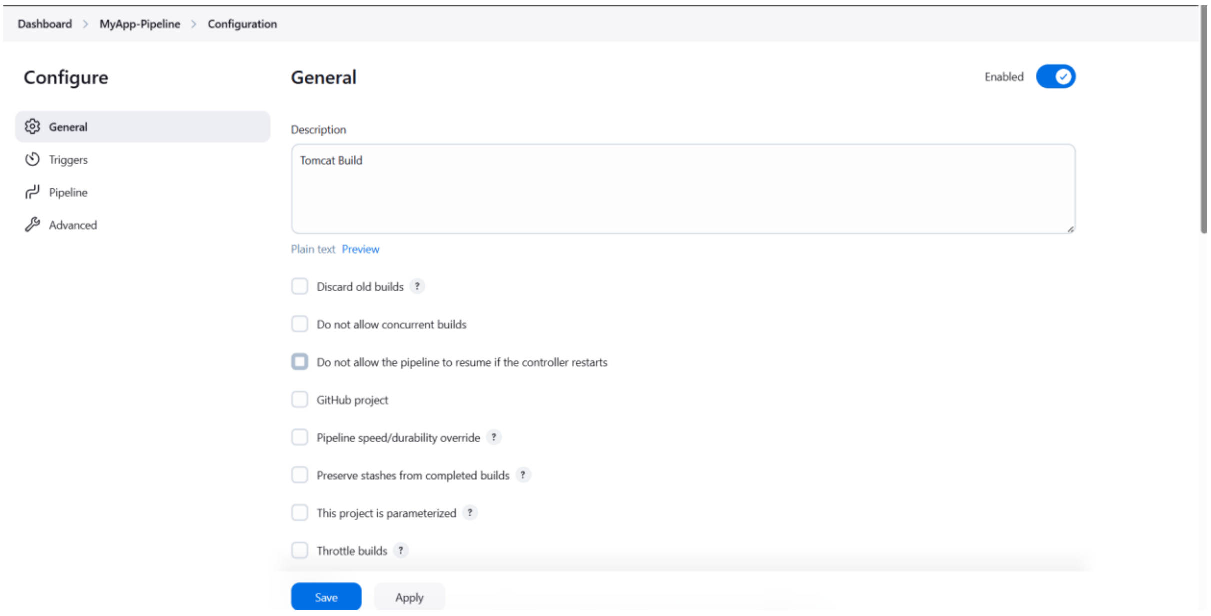
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.


Dashboard > MyApp-Pipeline > Configuration

Configure

General

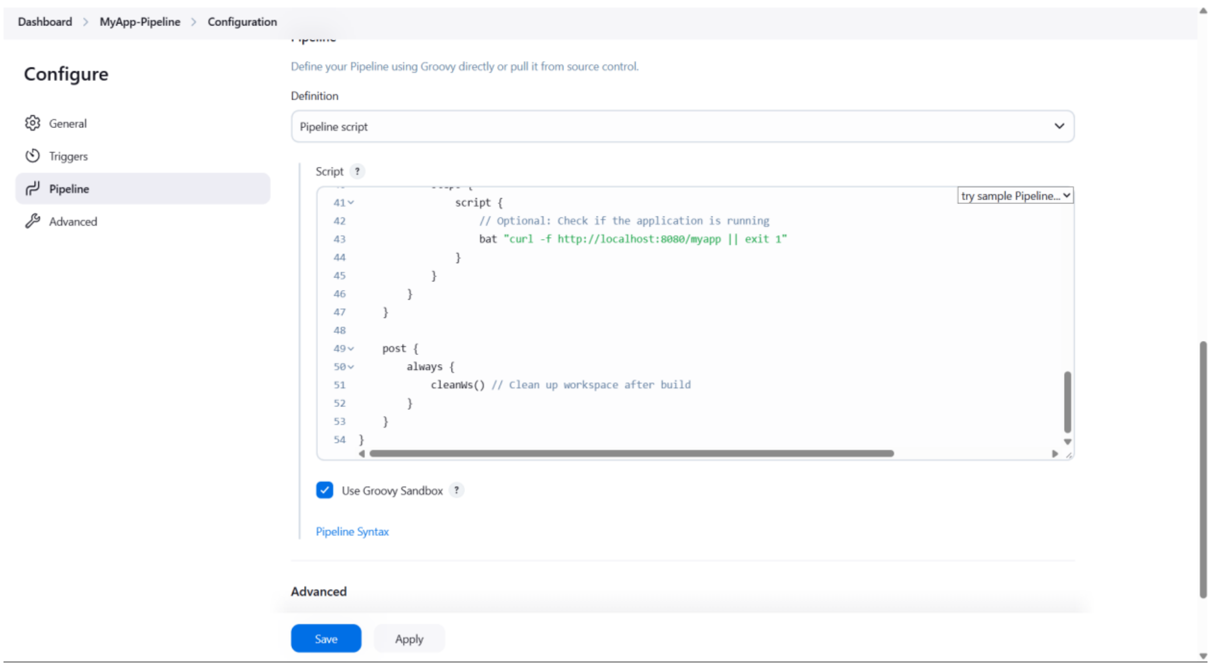
Enabled ✓

Description

Tomcat Build

Plain text Preview

- [ ] Discard old builds
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [ ] GitHub project
- [ ] Pipeline speed/durability override
- [ ] Preserve stashes from completed builds
- [ ] This project is parameterized
- [ ] Throttle builds


Dashboard > MyApp-Pipeline > Configuration

Configure

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script

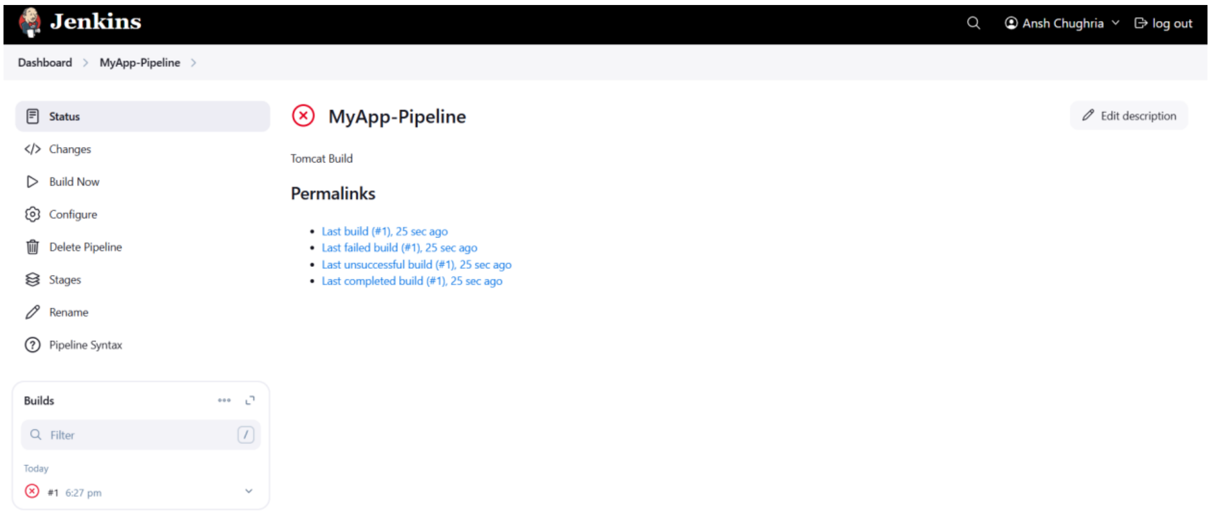
Script

```
                script {
                    // Optional: Check if the application is running
                    bat "curl -f http://localhost:8080/myapp || exit 1"
                }
            }
        }
    }

    post {
        always {
            cleanWs() // Clean up workspace after build
        }
    }
}
```

- [x] Use Groovy Sandbox

Pipeline Syntax

Advanced



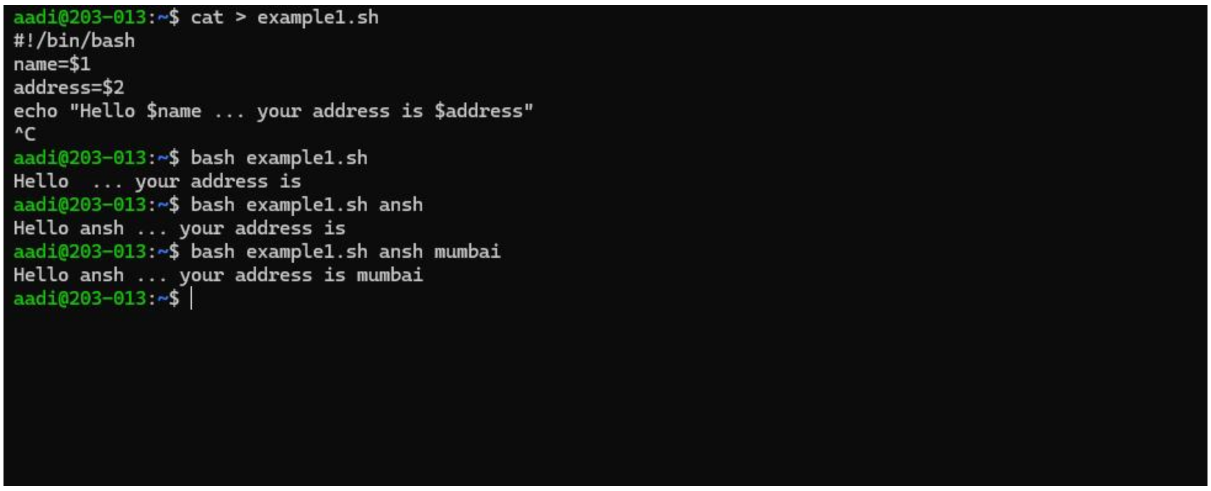
MyApp-Pipeline

Tomcat Build

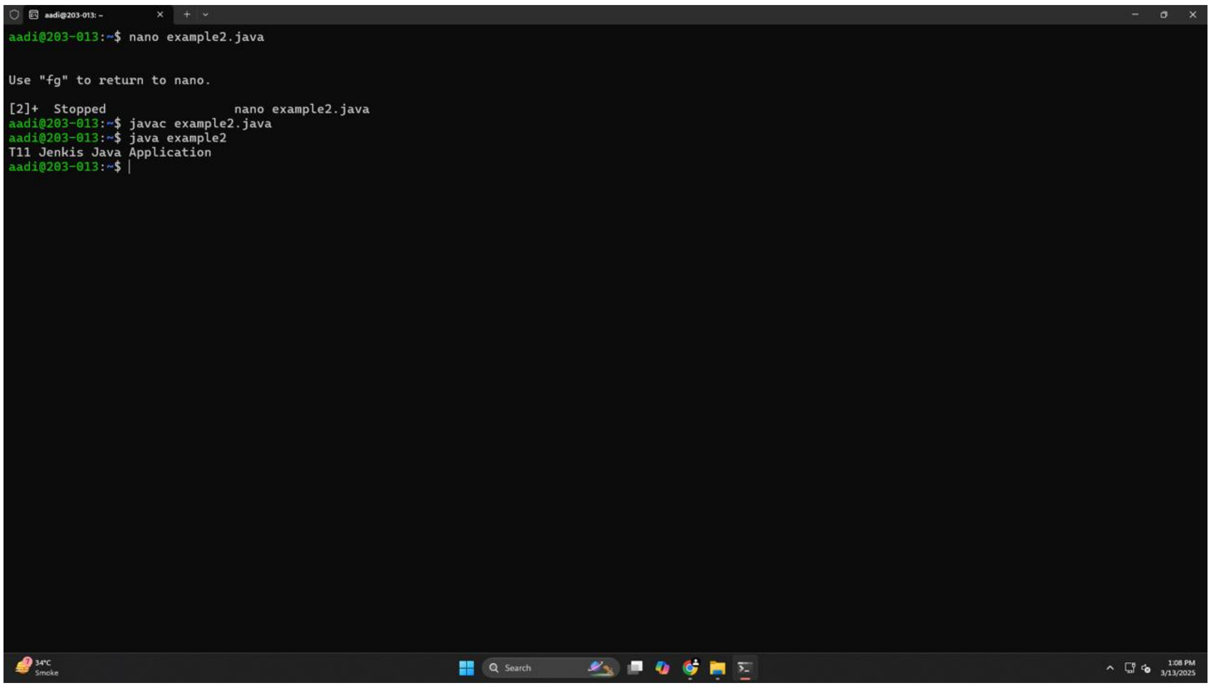
Permalinks

- Last build (#1), 25 sec ago
- Last failed build (#1), 25 sec ago
- Last unsuccessful build (#1), 25 sec ago
- Last completed build (#1), 25 sec ago

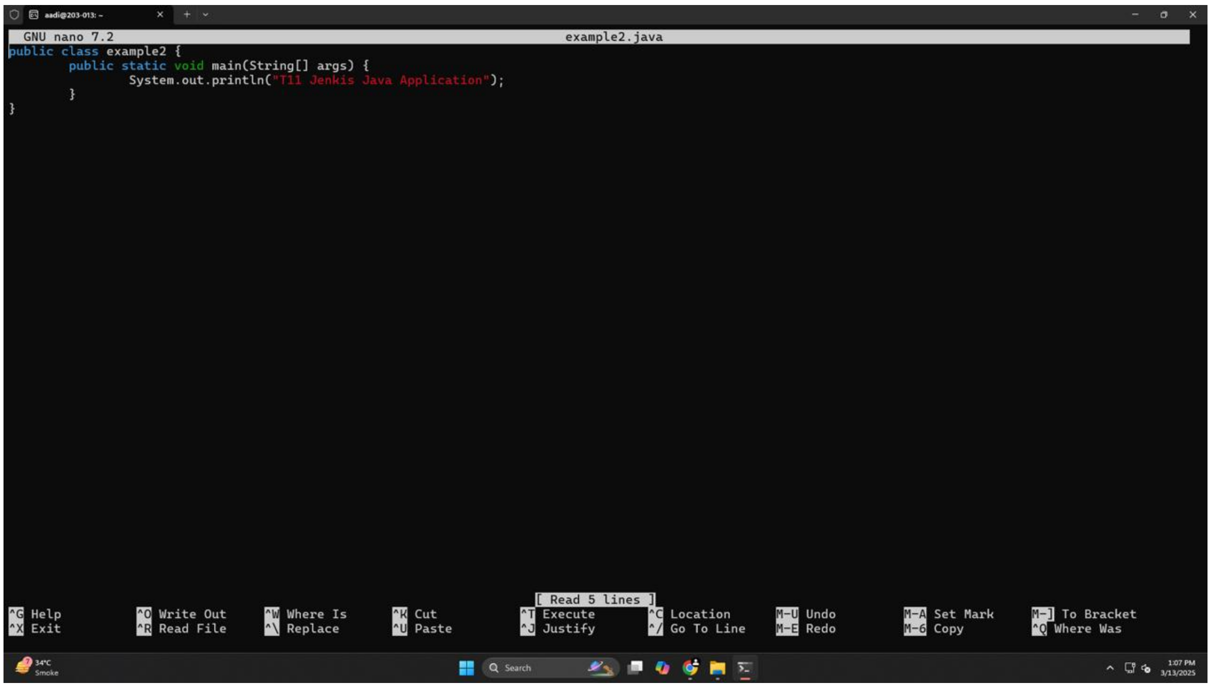
#1 6:27 pm



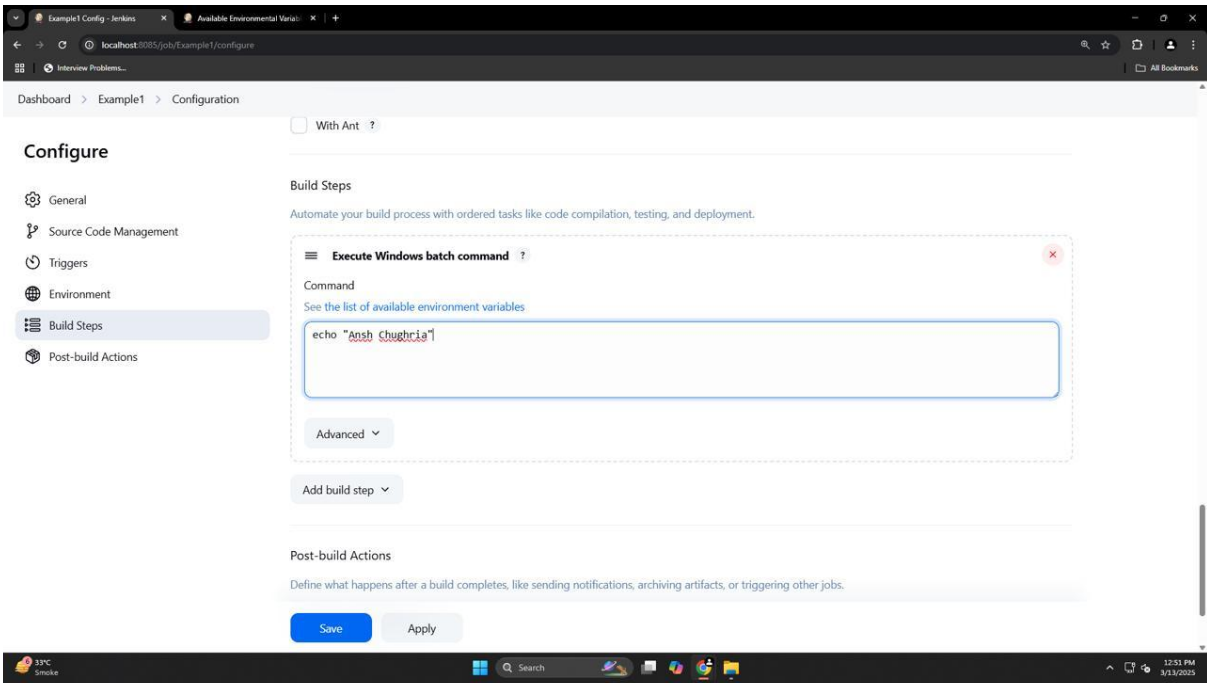
```
aadi@203-013:~$ cat > example1.sh
#!/bin/bash
name=$1
address=$2
echo "Hello $name ... your address is $address"
^C
aadi@203-013:~$ bash example1.sh
Hello  ... your address is
aadi@203-013:~$ bash example1.sh ansh
Hello ansh ... your address is
aadi@203-013:~$ bash example1.sh ansh mumbai
Hello ansh ... your address is mumbai
aadi@203-013:~$ |
```



```
aadi@203-013:~$ nano example2.java

Use "fg" to return to nano.

[2]+  Stopped                 nano example2.java
aadi@203-013:~$ javac example2.java
aadi@203-013:~$ java example2
T11 Jenkis Java Application
aadi@203-013:~$ |
```



```
  GNU nano 7.2                                   example2.java
public class example2 {
        public static void main(String[] args) {
                System.out.println("T11 Jenkis Java Application");
        }
}

                                         [ Read 5 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark   M-] To Bracket
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy       ^Q Where Was
```



Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

Execute Windows batch command

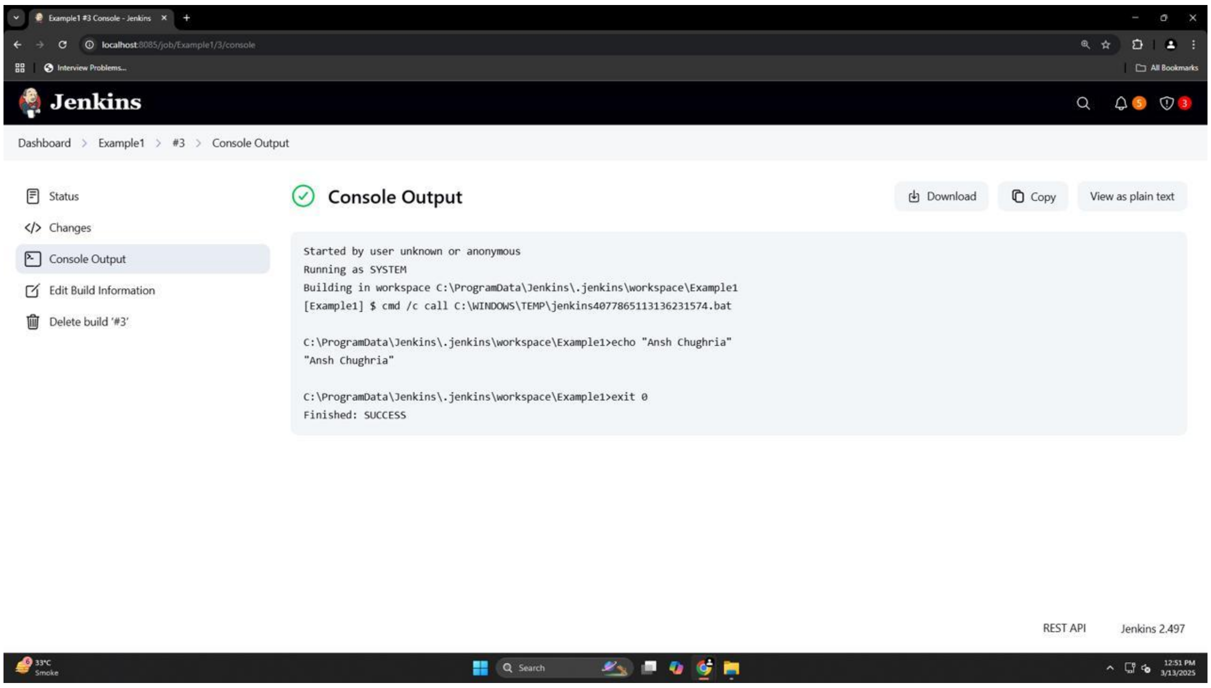
Command

See the list of available environment variables

```
echo "Ansh Chughria"
```

Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.



```
Started by user unknown or anonymous
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\.jenkins\workspace\Example1
[Example1] $ cmd /c call C:\WINDOWS\TEMP\jenkins4077865113136231574.bat

C:\ProgramData\Jenkins\.jenkins\workspace\Example1>echo "Ansh Chughria"
"Ansh Chughria"

C:\ProgramData\Jenkins\.jenkins\workspace\Example1>exit 0
Finished: SUCCESS
```

Conclusion:

This guide describes how to set up a **CI pipeline** in **Jenkins** that:

- Uses **Maven**, **Gradle**, or **Ant** to build and test an application.
- Automatically deploy the application to a **Tomcat server** once the build and tests are successful.

By using Jenkins pipelines, you can streamline the process of building, testing, and deploying software, ensuring quicker release cycles and fewer errors in production.

This process can be extended to support additional stages such as:

- **Code quality checks** (using tools like SonarQube).
- **Automated testing** for UI or API testing.
- **Notification of build status** via email or other integrations (e.g., Slack).